# A Dynamic Edit Distance Table*

Sung-Ryul Kim and Kunsoo Park

School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea
{kimsr,kpark}@theory.snu.ac.kr

**Abstract.** In this paper we consider the incremental/decremental version of the edit distance problem: given a solution to the edit distance between two strings $A$ and $B$, find a solution to the edit distance between $A$ and $B'$ where $B' = aB$ (incremental) or $bB' = B$ (decremental). As a solution for the edit distance between $A$ and $B$, we define the difference representation of the $D$-table, which leads to a simple and intuitive algorithm for the incremental/decremental edit distance problem.

## 1 Introduction

Given two strings $A[1..m]$ and $B[1..n]$ over an alphabet $\Sigma$, the *edit distance* between $A$ and $B$ is the minimum number of *edit operations* needed to convert $A$ to $B$. The edit distance problem is to find the edit distance between $A$ and $B$. Most common edit operations are the following.

1. *change*: replace one character of $A$ by another single character of $B$.
2. *deletion*: delete one character from $A$.
3. *insertion*: insert one character into $B$.

A well-known method for solving the edit distance problem in $O(mn)$ time uses the *D-table* [1,10]. Let $D(i,j)$, $0 \le i \le m$ and $0 \le j \le n$, be the edit distance between $A[1..i]$ and $B[1..j]$. Initially, $D(i,0) = i$ for $0 \le i \le m$ and $D(0,j) = j$ for $0 \le j \le n$. An entry $D(i,j)$, $1 \le i \le m$ and $1 \le j \le n$, of the $D$-table is determined by the three entries $D(i-1,j-1)$, $D(i-1,j)$, and $D(i,j-1)$. The recurrence for the $D$-table is as follows: For all $1 \le i \le m$ and $1 \le j \le n$,

$$D(i,j) = \min\{D(i-1,j-1) + \delta_{ij}, D(i-1,j) + 1, D(i,j-1) + 1\} \quad (1)$$

where $\delta_{ij} = 0$ if $A[i] = B[j]$; $\delta_{ij} = 1$, otherwise.

In this paper we consider the following incremental (resp. decremental) version of the edit distance problem: given a solution for the edit distance between $A$ and $B$, compute a solution for the edit distance between $A$ and $aB$ (resp. $B'$ where $B = bB'$), where $a$ (resp. $b$) is a symbol in $\Sigma$. By a *solution* we mean some

* This work was supported by the Brain Korea 21 Project.

R. Giancarlo and D. Sankoff (Eds.): CPM 2000, LNCS 1848, pp. 60–68, 2002.

encoding of the $D$-table computed between $A$ and $B$. Since essentially the same techniques can be used to solve both incremental and decremental versions of the edit distance problem, we will consider only the decremental version.

The incremental/decremental version of the edit distance problem was first considered by Landau et al. [3]. They used the $C$-table [2,4,5,7,9] (represented with linked lists) as a solution for the edit distance between $A$ and $B$. Given a threshold $k$ on the edit distance, their algorithm runs in $O(k)$ time. (If the threshold $k$ is not given, it runs in $O(m+n)$ time.) However, the result in [3] is quite complicated.

As a solution for the edit distance between $A$ and $B$, we define the difference representation of the $D$-table ($DR$-table for short). Each entry $DR(i,j)$ in the $DR$-table between $A$ and $B$ has two fields defined as follows: For $1 \leq i \leq m$ and $1 \leq j \leq n$,

1. $DR(i,j).U = D(i,j) - D(i-1,j)$
2. $DR(i,j).L = D(i,j) - D(i,j-1)$

A third field $DR(i,j).UL$, which is defined to be $D(i,j) - D(i-1,j-1)$, will be used later, but it need not be stored in $DR(i,j)$ because it can be computed as $DR(i,j).U + DR(i-1,j).L$. Because the possible values that each of $DR(i,j).U$ and $DR(i,j).L$ can have are $-1, 0$, and 1 [8], we need only four bits to store an entry in the $DR$-table. It is easy to see that the $D$-table can be converted to the $DR$-table in $O(mn)$ time, and vice versa. We can also compute one row (resp. column) of the $D$-table from the $DR$-table in $O(n)$ (resp. $O(m)$) time.

In this paper we present an $O(m+n)$-time algorithm for the incremental/decremental edit distance problem. Our result is much simpler and more intuitive than that of Landau et al. [3]. A key tool in our algorithm is the *change table* between the two $D$-tables before and after an increment/decrement. The change table is not actually constructed in our algorithm, but it is central in understanding our algorithm.

Our result finds a variety of applications. To verify whether a string $p$ is an approximate period of another string $x$ where $|x| = n$ and $|p| = m$, one needs to find the edit distance between $p$ and every substring of $x$ [6]. A naive method that computes a $D$-table of size $O(m^2)$ for each position of $x$ will take $O(m^2 n)$ time, but our algorithm reduces the time complexity to $O(mn)$ [6]. Other applications include the longest prefix match problem, the approximate overlap problem, the cyclic string comparison problem, and the text screen update problem [3].

This paper is organized as follows. In section 2, we describe the important properties of the change table. In section 3, we present our algorithm for the incremental/decremental edit distance problem.

## 2 Preliminary Properties

Let $\Sigma$ be a finite *alphabet* of *symbols*. A *string* over $\Sigma$ is a finite sequence of symbols in $\Sigma$. The length of a string $A$ is denoted by $|A|$. The $i$-th symbol in

$A$ is denoted by $A[i]$ and the substring consisting of the $i$-th through the $j$-th symbols of $A$ is denoted by $A[i..j]$.

Let $A$ and $B$ be strings of lengths $m$ and $n$, respectively, over $\Sigma$, and let $B' = B[2..n]$. Let $D$ be the $D$-table between $A$ and $B$ and let $D'$ be the $D$-table between $A$ and $B'$. Also let $DR$ be the $DR$-table between $A$ and $B$ and let $DR'$ be the $DR$-table between $A$ and $B'$. In this section, we prove the key properties between $D$ and $D'$ that enables us to compute efficiently $DR'$ from $DR$.

***D***

| | | *b* | *b* | *a* | *b* | *a* | *b* | *b* | *a* | *b* |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| *a* | 1 | 1 | 2 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| *b* | 2 | 1 | 1 | 2 | 2 | 3 | 4 | 5 | 6 | 7 |
| *a* | 3 | 2 | 2 | 1 | 2 | 2 | 3 | 4 | 5 | 6 |
| *b* | 4 | 3 | 2 | 2 | 1 | 2 | 2 | 3 | 4 | 5 |
| *b* | 5 | 4 | 3 | 3 | 2 | 2 | 2 | 2 | 3 | 4 |
| *a* | 6 | 5 | 4 | 3 | 3 | 2 | 3 | 3 | 2 | 3 |
| *b* | 7 | 6 | 5 | 4 | 3 | 3 | 2 | 3 | 3 | 2 |
| *b* | 8 | 7 | 6 | 5 | 4 | 4 | 3 | 2 | 3 | 3 |

***D'***

| | | | *b* | *a* | *b* | *a* | *b* | *b* | *a* | *b* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| *a* | | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| *b* | | 2 | 1 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| *a* | | 3 | 2 | 1 | 2 | 1 | 2 | 3 | 4 | 5 |
| *b* | | 4 | 3 | 2 | 1 | 2 | 1 | 2 | 3 | 4 |
| *b* | | 5 | 4 | 3 | 2 | 2 | 2 | 1 | 2 | 3 |
| *a* | | 6 | 5 | 4 | 3 | 2 | 3 | 2 | 1 | 2 |
| *b* | | 7 | 6 | 5 | 4 | 3 | 2 | 3 | 2 | 1 |
| *b* | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 3 | 2 |

***Ch***

| | | | *b* | *a* | *b* | *a* | *b* | *b* | *a* | *b* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| *a* | | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| *b* | | 1 | 0 | 0 | -1 | -1 | -1 | -1 | -1 | -1 |
| *a* | | 1 | 0 | 0 | 0 | -1 | -1 | -1 | -1 | -1 |
| *b* | | 1 | 1 | 0 | 0 | 0 | -1 | -1 | -1 | -1 |
| *b* | | 1 | 1 | 0 | 0 | 0 | 0 | -1 | -1 | -1 |
| *a* | | 1 | 1 | 1 | 0 | 0 | 0 | -1 | -1 | -1 |
| *b* | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | -1 | -1 |
| *b* | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | -1 |

**Fig. 1.** An example $Ch$-table

One key tool in understanding our algorithm is the *change table* ($Ch$-table for short) from $D$ to $D'$. Later, when we compute $DR'$ from $DR$, the first column of $DR$ is discarded and each entry $DR(i, j+1)$, $0 \le i \le m$ and $0 \le j < n$, will be converted to $DR'(i, j)$. Thus, each entry in the $Ch$-table $Ch$ from $D$ to $D'$ is defined as follows:

$$Ch(i, j) = D'(i, j) - D(i, j+1).$$

The $Ch$-table is not actually constructed in our algorithm because the initialization of the $Ch$-table will require $\Theta(mn)$ time. It will be used only for the description of the algorithm. See Fig. 1 for an example $Ch$-table.

Figure 1 suggests a property of the $Ch$-table: the entries of value $-1$ (resp. 1) appear contiguously in the upper-right (resp. lower-left) part of the $Ch$-table in a *staircase-shaped* region. This property is formally proved in the following series of lemmas.

**Lemma 1.** *In the $Ch$-table $Ch$, the following properties hold.*

1. $Ch(0, j) = -1$ *for all* $0 \le j < n$.
2. $Ch(i, 0) = 0$ *for all* $1 \le i < k$, *where* $k$ *is the smallest index in* $A$ *such that* $A[k] = B[1]$.
3. $Ch(i, 0) = 1$ *for all* $k \le i \le m$.

*Proof.* Immediate from the definition of the $D$-table.

**Lemma 2.** *For* $1 \le i \le m$ *and* $1 \le j < n$, *the possible values of* $Ch(i, j)$ *are in the range* $\min\{Ch(i-1, j-1), Ch(i-1, j), Ch(i, j-1)\}..\max\{Ch(i-1, j-1), Ch(i-1, j), Ch(i, j-1)\}$.

*Proof.* Recall that $Ch(i, j)$ is defined to be $D'(i, j) - D(i, j + 1)$. By recurrence (1), $D(i, j + 1)$ is

$$\min\{D(i-1, j) + \delta_{i,j+1}, D(i-1, j+1) + 1, D(i, j) + 1\}. \quad (2)$$

Also, $D'(i, j)$ is $\min\{D'(i-1, j-1) + \delta'_{ij}, D'(i-1, j) + 1, D'(i, j-1) + 1\}$ where $\delta'_{ij} = 0$ if $A[i] = B'[j]$; $\delta'_{ij} = 1$, otherwise. Because $B'[j]$ is the same symbol as $B[j + 1]$, $\delta'_{ij} = \delta_{i,j+1}$. Hence,

$$D'(i, j) = \min \begin{cases} D(i-1, j) + Ch(i-1, j-1) + \delta_{i,j+1} \\ D(i-1, j+1) + Ch(i-1, j) + 1 \\ D(i, j) + Ch(i, j-1) + 1. \end{cases} \quad (3)$$

Note that the only differences between (2) and (3) are additional terms $Ch(i-1, j-1), Ch(i-1, j)$, and $Ch(i, j-1)$ in (3). Assume without loss of generality that the second argument is minimum in (2). If the second argument is minimum in (3), the lemma holds because $Ch(i, j) = Ch(i-1, j)$. Otherwise, assume without loss of generality that the third argument is minimum in (3). Then $Ch(i, j) = D(i, j) + Ch(i, j-1) + 1 - (D(i-1, j+1) + 1)) \geq Ch(i, j-1)$ because the second argument is minimum in (2). Also, $Ch(i, j) \leq Ch(i-1, j)$ because the third argument is minimum in (3).

**Corollary 1.** *The possible values of* $Ch(i, j)$ *are* $-1, 0,$ *and* $1$.

*Proof.* It follows from Lemmas 1 and 2.

**Lemma 3.** *For each* $0 \leq i \leq m$*, let* $f(i)$ *be the smallest integer* $j$ *such that* $Ch(i, j) = -1$. *(*$f(i) = n$ *if* $Ch(i, j') \neq -1$ *for* $0 \leq j' < n$*.) Then,* $Ch(i, j') = -1$ *for all* $f(i) \leq j' < n$*. Furthermore,* $f(i) \geq f(i-1)$ *for* $1 \leq i \leq m$*.*

*Proof.* We use induction on $i$. When $i = 0$, $f(i) = 0$ and the lemma holds by Lemma 1. Assume inductively that the lemma holds for $i = k$. That is, $Ch(k, j') \neq -1$ for $0 \leq j' < f(k)$ and $Ch(k, j') = -1$ for $f(k) \leq j < n$.

Let $Ch(k+1, l)$ be the first entry in row $k+1$ that is $-1$. For $Ch(k+1, l)$ to be $-1$, at least one of $Ch(k, l-1)$ and $Ch(k, l)$ must be $-1$ by Lemma 2. Thus, we have shown that $l = f(k+1) \geq f(k)$. It is easy to see that $Ch(k+1, l') = -1$ for $f(k+1) < l' < n$ by the inductive assumption, the condition that $f(k+1) \geq f(k)$, and Lemma 2.

The following lemma is symmetric to Lemma 3 and it can be similarly proved.

**Lemma 4.** *For each* $0 \leq j < n$*, let* $g(j)$ *be the smallest integer* $i$ *such that* $Ch(i, j) = 1$. *(*$g(j) = m + 1$ *if* $Ch(i', j) \neq 1$ *for* $0 \leq i' \leq m$*.) Then,* $Ch(i', j) = 1$ *for all* $g(j) \leq i' \leq m$*. Furthermore,* $g(j) \geq g(j-1)$ *for* $1 \leq j < n$*.*

We say that an entry $Ch(i, j)$ is *affected* if the values of $Ch(i-1, j-1), Ch(i-1, j)$, and $Ch(i, j-1)$ are not the same. We also say that $DR'(i, j)$ is affected if $Ch(i, j)$ is affected.

**Lemma 5.** *If $DR'(i,j)$ is not affected, then $DR'(i,j)$ equals $DR(i,j+1)$.*

*Proof.* If $DR'(i,j)$ is not affected, then the value of $Ch(i,j)$ is the same as the common value of $Ch(i-1,j-1), Ch(i-1,j)$, and $Ch(i,j-1)$ by Lemma 2. Then $DR'(i,j).U = D'(i,j) - D'(i-1,j) = D(i,j+1) + Ch(i,j) - (D(i-1,j+1) + Ch(i-1,j)) = DR(i,j+1).U$. Similarly, $DR'(i,j).L = DR(i,j+1).L$.

We say that an entry $Ch(i,j)$ is a $(-1)$-*boundary* (resp. 1-*boundary*) *entry* if $Ch(i,j)$ is of value $-1$ (resp. 1) and at least one of $Ch(i,j-1), Ch(i+1,j)$, and $Ch(i+1,j-1)$ (resp. $Ch(i,j+1), Ch(i-1,j)$, and $Ch(i-1,j+1)$) is not of value $-1$ (resp. 1).

By Lemma 5 we can conclude that in computing $DR'$ from $DR$, only the affected entries need be changed. See Fig. 1 again. Because the entries whose values are $-1$ (or 1) appear contiguously in the $Ch$-table, the affected entries are either $(-1)$- or 1-boundary entries themselves or appear adjacent to $(-1)$- or 1-boundary entries. The key idea of our algorithm is to scan the $(-1)$- and 1-boundary entries starting from the upper-left corner of the $DR$-table when we compute the affected entries. Lemmas 3 and 4 imply that the number of $(-1)$- and 1-boundary entries in the $DR$-table is $O(m+n)$.

## 3 Boundary Scan Algorithm

In this section we show how to compute $DR'$ from $DR$. First, we describe how we scan the boundary entries starting from the upper-left corner of the $DR'$-table within the proposed time complexity. Then, we will mention the modifications to the boundary-scan algorithm which leads to an algorithm that converts $DR$ to $DR'$.

For simplicity we will use the $Ch$-table in the description of our algorithm. However, the $Ch$-table is not explicitly constructed but accessed through the one-dimensional tables $f()$ and $g()$. The details will be given later.

**Lemma 6.**

$$Ch(i,j) = \min \begin{cases} -DR(i,j+1).UL + Ch(i-1,j-1) + \delta_{i,j+1} \\ -DR(i,j+1).U + Ch(i-1,j) + 1 \\ -DR(i,j+1).L + Ch(i,j-1) + 1 \end{cases}$$

*(i.e., $Ch(i-1,j-1), Ch(i-1,j), Ch(i,j-1)$, and $DR(i,j+1)$ are needed to compute $Ch(i,j)$).*

*Proof.* Recall that $Ch(i,j) = D'(i,j) - D(i,j+1)$. Substituting recurrence (1) for $D'(i,j)$ and distributing $D(i,j+1)$ into the min function, we have $Ch(i,j) = \min\{\ldots, D'(i-1,j) - D(i,j+1) + 1, \ldots\}$ (only the second argument is shown). Substituting $D(i-1,j+1) + Ch(i-1,j)$ for $D'(i-1,j)$, the second argument becomes $D(i-1,j+1) - D(i,j+1) + Ch(i-1,j) + 1 = -DR(i,j+1).U + Ch(i-1,j) + 1$. The lemma follows from similar calculations for the first and the third arguments.

**Algorithm 1**

Let $k$ be the smallest index in $A$ such that $A[k] = B[1]$.
$(i_{-1}, j_{-1}) \leftarrow (0, 1)$; $(i_1, j_1) \leftarrow (k, 0)$; $f(0) \leftarrow 0$; $g(0) \leftarrow k$
$finished_{-1} \leftarrow$ **false**
$finished_1 \leftarrow$ **false**
**while not** $finished_{-1}$ **or not** $finished_1$ **do**
  **if** $i_{-1} < i_1 - 1$ **then** {Case 1}
    Compute $Ch(i_{-1} + 1, j_{-1})$. {See Fig. 4.}
    **if** $Ch(i_{-1} + 1, j_{-1}) = -1$ **then**
      $i_{-1} \leftarrow i_{-1} + 1$; $f(i_{-1}) \leftarrow j_{-1}$
    **else**
      $j_{-1} \leftarrow j_{-1} + 1$
    **fi**
  **else if** $j_1 < j_{-1} - 1$ **then** {Case 2}
    Symmetric to Case 1.
  **else** {Case 3, $i_1 = i_{-1} + 1$ and $j_1 = j_{-1} - 1$ }
    Compute $Ch(i_{-1} + 1, j_{-1})$. {See Fig. 5.}
    **if** $Ch(i_{-1} + 1, j_{-1}) = -1$ **then**
      $i_{-1} \leftarrow i_{-1} + 1$; $i_1 \leftarrow i_1 + 1$; $f(i_{-1}) \leftarrow j_{-1}$
    **else if** $Ch(i_{-1} + 1, j_{-1}) = 1$ **then**
      $j_{-1} \leftarrow j_{-1} + 1$; $j_1 \leftarrow j_1 + 1$; $g(j_1) \leftarrow i_1$
    **else**
      $j_{-1} \leftarrow j_{-1} + 1$; $i_1 \leftarrow i_1 + 1$
    **fi**
  **fi**
  **if** $i_{-1} = m$ or $j_{-1} = n$ **then** $finished_{-1} \leftarrow$ **true fi**
  **if** $i_1 = m + 1$ or $j_1 = n - 1$ **then** $finished_1 \leftarrow$ **true fi**
**od**

**Fig. 2.** Algorithm 1

Algorithm 1 is the boundary-scan algorithm. In the algorithm, the pair $(i_{-1}, j_{-1})$ (resp. $(i_1, j_1)$) indicates that $Ch(i_{-1}, j_{-1})$ (resp. $Ch(i_1, j_1)$) is the current $(-1)$-boundary (resp. 1-boundary) entry that is being scanned. The following property holds for $Ch(i_{-1}, j_{-1})$ and $Ch(i_1, j_1)$ by Lemmas 3 and 4. See Fig. 3 for an illustration.

*Property 1.*

1. $Ch(i, j) \neq -1$ if $i > i_{-1}$ and $j < j_{-1}$.
2. $Ch(i, j) \neq 1$ if $i < i_1$ and $j > j_1$.

In one iteration of the loop in Algorithm 1, one or both of the current boundary entries are moved to the next boundary entries. For example, the current $(-1)$-boundary entry is moved to the next $(-1)$-boundary entry which can be down or to the right of the current $(-1)$-boundary entry. We maintain the following invariants in each iteration of Algorithm 1.

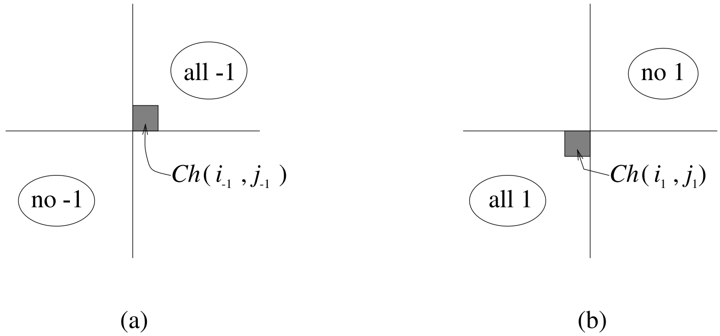


**Fig. 3.** Boundary entry conditions

**Invariant 1**

1. *$i_{-1} < i_1$ and $j_{-1} > j_1$.*
2. *All values of $f(0), \ldots, f(i_{-1})$ are known.*
3. *All values of $g(0), \ldots, g(j_1)$ are known.*

One iteration of Algorithm 1 has three cases. Case 1 applies when the current $(-1)$-boundary can be moved by one entry (down or to the right) without violating Invariant 1.1. Case 2 applies when the current 1-boundary can be moved by one entry (down or to the right) without violating Invariant 1.1. Case 3 applies when moving the $(-1)$-boundary entry down by one entry or moving the 1-boundary entry to the right by one entry will violate Invariant 1.1, and thus both boundary entries have to be moved simultaneously. What Algorithm 1 does in each case is described in Fig. 2.

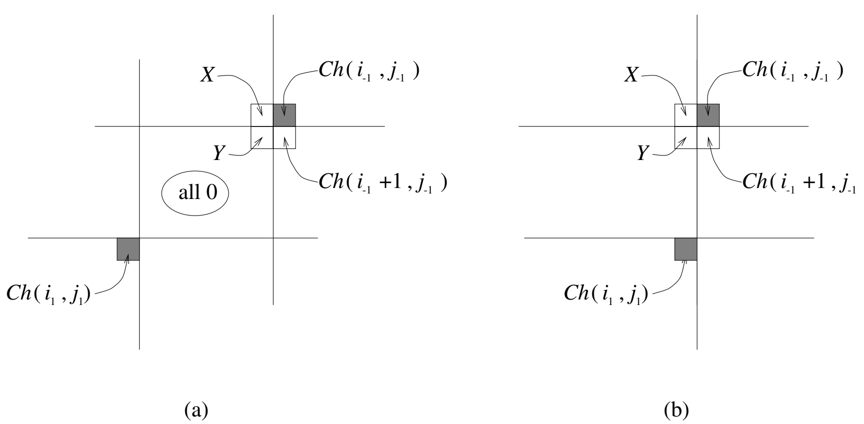


**Fig. 4.** Case 1

What remains to show is the methods to obtain the values of the $Ch$-table entries that are used to compute a new $Ch$-table entry, e.g., $Ch(i_{-1}+1, j_{-1})$ in Case 1. The two subcases for Case 1 are depicted in Fig. 4. The first subcase is when $j_{-1} > j_1 + 1$. See Fig. 4 (a). The unknown values of the $Ch$-table entries are $X$ and $Y$. By Invariant 1.2 the value of $f(i_{-1})$ is known. If $f(i_{-1}) < j_{-1}$, then $X = -1$. Otherwise ($f(i_{-1}) = j_{-1}$), $X = 0$ because $X$ is not 1 by Property 1.1. It is easy to see that $Y = 0$ because $Y$ is inside the region in which there are no $(-1)$'s (by Property 1.1) and no 1's (by Property 1.2). The second subcase is when $j_{-1} = j_1 + 1$. See Fig. 4 (b). We can compute the value of $X$ as $-1$ if $f(i_{-1}) < j_{-1}$; 1 if $g(j_1) \le i_{-1}$; 0, otherwise. We know that $Y \neq -1$ by Property 1.1. Thus, $Y = 1$ if $g(j_1) \le i_{-1} + 1$; $Y = 0$, otherwise. Case 3 is depicted in Fig. 5. The value of $X$ can be computed as we computed the value of $X$ in the second subcase of Case 1.

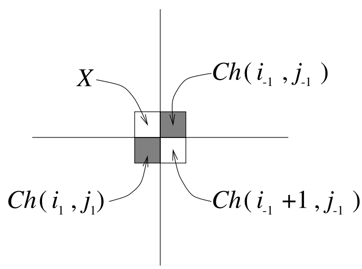


**Fig. 5.** Case 3

We now show that all affected $Ch$-table entries are computed by Algorithm 1. It is easy to see that each affected entry $Ch(i, j)$, $1 \le i \le m$ and $1 \le j < n$, falls into one of the following types by Lemmas 3 and 4. For each of the types we can easily check which cases in our algorithm compute $Ch(i, j)$.

1. $Ch(i, j)$ is a $(-1)$-boundary entry such that $Ch(i, j-1) \neq -1$: $Ch(i, j)$ is computed by Case 1 if $Ch(i, j-1) = 0$; by Case 3, otherwise.
2. $Ch(i, j)$ is an 1-boundary entry such that $Ch(i-1, j) \neq 1$: $Ch(i, j)$ is computed by Case 2 if $Ch(i-1, j) = 0$; by Case 3, otherwise.
3. $Ch(i, j) = 0$ and either $Ch(i-1, j) = -1$ or $Ch(i, j-1) = 1$: $Ch(i, j)$ is computed by Case 1 if $Ch(i, j-1) = 0$; by Case 2 if $Ch(i-1, j) = 0$; by Case 3, otherwise.

To compute $DR'$ from $DR$, we first discard the first column from $DR$. Then, we run a modified version of Algorithm 1. The modifications to Algorithm 1 is to compute $DR'(i, j)$ whenever we compute the value of $Ch(i, j)$. Once $Ch(i, j)$ is computed using Lemma 6, the fields in $DR'(i, j)$ can be easily computed. That is, $DR'(i, j).L = DR(i, j+1).L + Ch(i, j) - Ch(i, j-1)$ and $DR'(i, j).U = DR(i, j+1).U + Ch(i, j) - Ch(i-1, j)$.

We can easily check that one iteration of the loop takes only constant time and that it increases at least one of $i_{-1}, j_{-1}, i_1, j_1$ by one. Hence, the time complexity of our algorithm is $O(m+n)$.

**Theorem 1.** *Let $A$ and $B$ be two strings of lengths $m$ and $n$, respectively, and $B' = B[2..n]$. Given the difference representation $DR$ between $A$ and $B$, the difference representation $DR'$ between $A$ and $B'$ can be computed in $O(m+n)$ time.*

## References


1. Galil, Z. and Giancarlo, R.: Data Structures and Algorithms for Approximate String Matching. J. Complexity 4 (1988) 33–72
2. Galil, Z. and Park, K.: An Improved Algorithm for Approximate String Matching. SIAM J. Computing, Vol. 19, No. 6 (1990) 989–999
3. Landau, G. M., Myers, E. W., and Schmidt, J. P.: Incremental String Comparison. SIAM J. Computing, Vol. 27, No. 2 (1998) 557–582
4. Landau, G. M. and Vishkin, U.: Fast String Matching with $k$ Differences. J. Comput. System Sci., 37 (1998) 63–78
5. Landau, G. M. and Vishkin, U.: Fast Parallel and Serial Approximate String Matching. J. Algorithms, 10 (1989) 157–169
6. Sim, J. S., Iliopoulos, C. S., Park, K., and Smyth, W. F.: Approximate Periods of Strings. In 10th Annual Symposium, Combinatorial Pattern Matching '99 (1999) 123–133
7. Ukkonen, E.: Algorithms for Approximate String Matching. Inform. and Control, 64 (1985) 100–118
8. Ukkonen, E.: Finding Approximate Patterns in Strings. J. of Algorithms, 6 (1985) 132–137
9. Ukkonen, E. and Wood, D.: Approximate String Matching with Suffix Automata. Algorithmica, 10 (1993) 353–364
10. Wagner, R. A. and Fisher, M. J.: The String-to-string Correction Problem. J. Assoc. Comput. Mach., 21 (1974) 168–173